CMPS 111 - Spring 2002
Final Exam
June 6, 2002

Name:_____ID:_____

This is a closed note, closed book exam. There are 40 multiple choice questions and 12 short answer questions. Plan your time accordingly.

Part I: Multiple Choice - Write the letter of the best answer to the left of each question (2 points each)

1. Which is false:
   a. Mainframe computers differ greatly from PCs in terms of their I/O capacity
   b. Most PC operating systems do not support multiprogramming
   c. The main difference between real-time and general purpose operating systems is the scheduling

2. System calls:
   a. Provide a rich and flexible API for software developers to use
   b. Protect important kernel data structures from user code
   c. Increase the performance of the operating system

Oops, none of these is true

3. Which is true about process termination:
   a. The waitpid operation always blocks the calling process
   b. A child process can kill another child process of the same parent process
   c. A child process that exits before the parent process does a waitpid operation becomes a zombie process

4. Which is true about processes and threads:
   a. Threads in a process share the same stack
   b. Threads in a process share the same file descriptors
   c. Threads in a process share the same register values

Oops, none of these is true

5. In the process state transition diagram, the transition from the Ready state to the Running state:
   a. Indicates that a process was preempted by another process
   b. Indicates that a process has blocked on an I/O operation
   c. Indicates that a process is done waiting for an I/O operation

6. In a user-level thread package, threads are scheduled by:
   a. The kernel
   b. The user
   c. The process

7. A critical region is:
   a. The part of a program in which shared data is accessed.
   b. The most important part of a program
   c. The part of the kernel that interfaces to the device controllers

8. Semaphores are used for:
   a. Resource abstraction and resource sharing
   b. Data hiding and encapsulation
   c. Mutual exclusion and synchronization

9. Which is true:
   a. Busy waiting with test-and-set is not always the least efficient method of implementing mutual exclusion
   b. When used for mutual exclusion, a semaphore should be initially set to 0
   c. Monitors provide synchronization and condition variables provide mutual exclusion

10. The Dining Philosophers is an example of:
    a. A bounded buffer problem
    b. A synchronization problem
    c. A readers/writers problem

11. Which are all goals of various CPU scheduling algorithms:
    a. Fairness, Throughput, and I/O Bandwidth
    b. Responsiveness, Utilization, and Timeliness
    c. Turnaround time, No Starvation, and First-Come-First-Serve

12. Which is true:
    a. Shortest Job Next is the best preemptive scheduling algorithm in terms of turnaround time
    b. First-Come-First-Serve can suffer from starvation
    c. Priority scheduling is the most general scheduling algorithm

13. Which is true:
    a. Bitmapped free memory management is fastest
    b. Linked list free memory management is the least memory efficient
    c. Bitmapped free memory management makes it easier to find contiguous free blocks

14. Which are advantages of paging:
    a. Efficient use of memory and higher degree of multiprogramming
    b. No overhead and doesn't require any hardware support
    c. Higher degree of multprogramming and no overhead

15. TLBs:
    a. Are used to cache frequently used data
    b. Are used to cache process register information
    c. Are used to cache page translation information

16. Inverted page tables are:
    a. Faster than regular page tables
    b. Smaller than regular page tables
    c. More flexible than regular page tables

17. Suppose you have a memory access stream that references pages 0, 1, and 2 over and over again, in that order, and you have only 2 pages of physical memory for the process, which page replacement algorithm will generate the fewest page faults
    a. Least Recently Used (LRU)
    b. First In First Out (FIFO)
    c. Most Recently Used (MRU)

18. Which is generally the best algorithm for managing the buffer cache:
    a. Least Recently Used (LRU)
    b. First In First Out (FIFO)
    c. Most Recently Used (MRU)

19. The working set of a process:
    a. Is statically determined based on the code
    b. Is impossible to determine at runtime
    c. Can change as the process executes

20. Belady's anomaly:
    a. Only affects stack algorithms
    b. Doesn't affect stack algorithms
    c. Has nothing to do with stack algorithms

21. The main difference between paging and segmentation is:
    a. The size of the objects being managed
    b. One refers to memory and the other refers to disk
    c. Nothing; they are two names for the same thing

22. The main advantage of DMA is that it:
    a. Increases the speed of the CPU
    b. Increases the speed of the data bus
    c. Increases the performance of the system by allowing more things to happen at once

23. Interrupts:
    a. Allow the CPU to notify devices that it needs attention
    b. Allow the devices to notify the device controller that they need attention
    c. Allow the devices to notify the CPU that they need attention

24. Buffering is useful because:
    a. It makes it seem like there is more memory in the machine
    b. It allows devices and the CPU to operate asynchronously
    c. It requires fewer memory copies

25. Typical hard drives have seek times of about:
    a. 1 millisecond
    b. 10 milliseconds
    c. 100 milliseconds
    d. 1 second

26. Typical hard drives have throughput of about:
    a. 1 megabytes per second
    b. 10 megabytes per second
    c. 100 megabytes per second
    d. 1 gigabytes per second

27. RAID is a way to:
    a. Increase hard drive performance and reliability
    b. Increase hard drive latency and throughput
    c. Increase hard drive throughput and decrease cost

28. Disk request scheduling algorithms attempt to minimize:
    a. Seek time
    b. Rotational latency
    c. Transfer time

29. The main purpose of directories is:
    a. User convenience
    b. To increase file system performance
    c. To increase file system capacity

30. The three general goals of computer security are:
    a. Detection, response, and correction
    b. Confidentiality, integrity, and availability
    c. Performance, reliability, and availability

31. Which is false:
    a. Shared-key cryptography stll requires that some information be sent unencrypted
    b. Public-key cryptography doesn't employ a secret key
    c. Public-key cryptography can also be used to solve the authentication problem

32. A Trojan Horse is:
    a. A special password that bypasses the regular password system
    b. A kind of computer virus
    c. A program that does something else in addition to its apparent purpose

33. The Unix file system uses a form of which protection mechanism:
    a. Protection Domains
    b. Access Control Lists
    c. Capabilities

34. In DLXOS or UNIX, how many systems calls must a process make to open a file, read three blocks of data from it, and close it?
    a. One
    b. Three
    c. Five

35. What is the main difference between traps and interrupts?
    a. How they are initiated
    b. What is done when they occur
    c. What happens after they are handled

36. Lottery Scheduling
    a. Provides strong performance guarantees
    b. Provides no real guarantees but has reasonable average case performance
    c. Provides absolutely random performance which cannot be characterized at all

Oops, none of these is true, it's 64K

37. Given a disk block size of 512 bytes and a disk with 10 GB of data (using 32 bit disk addresses), how big can a file be if you have a single index block in an indexed file management scheme?
    a. 512 bytes
    b. 4 MB
    c. 8 MB

38. The DLX code uses which scheduling algorithm by default?
    a. Round Robin
    b. Lottery
    c. First Come First Served

39. The DLXOS system call Spawn() returns:
    a. The exit status of the child process
    b. A pointer to the PCB of the child process
    c. The process ID of the child process

40. [Everyone gets credit for this one] In retrospect, DLXOS was:
    a. A good learning tool
    b. A poor learning tool
    c. I refuse to answer on the grounds that Prof. Miller might see my answer and try to keep me from graduating

Part II: Short answers (10 points each)
Note: You may designate one of these questions to be counted as Extra Credit.

1. Processes and threads

a) Explain what a process is and what information the OS keeps about each process.

A process is an executing program. The OS keeps track of three main things: process status (Program Counter, Stack Pointer, etc.), address space information (code location, limits, protection, virtual memory), and information about open files (file descriptors).

b) In this context, explain what a thread is. Be clear about the differences between threads and processes.

In a multithreaded environment, a thread is a unit of execution within a process. It contains its own process status information but shares the address space and file descriptor information with other threads in the same process. In such an environment ultiple threads can be executing different portions of the code in the same process, and a process is just a context in which the execution takes place. By contrast, in a nonthreaded environment as described in part (a), a process contains all three parts and has only one unit of execution.

c) Explain the advantage of using multiple threads instead of multiple processes to solve some programming problems.

Threads allow for efficient sharing of information, since the address space is shared, multiple threads can directly access the same memory.

d) What is the difference between user threads and kernel threads? Why would you ever choose to use user threads?

Kernel threads are scheduled by the kernel, whereas user threads are scheduled by the process, which is in turn scheduled by the kernel.

2. Explain system calls in a way that would be understandable to someone who has not yet taken CMPS 111. For example: What are they? Why do we need them? What do they do? How do the work? Etc.

System calls are an API to the functionality of the operating system. Operating systems manage the system resources and support sharing of critical resources and provide convenient abstractions on top of the basic hardware. As part of this, the operating system must provide certain guarantees allowing some users to access some data and not others. But to do their work, the OS itself needs to access all data. In addition, the kernel itself maintains internal data structures that must be kept consistent and, in some cases, private in order to guarantee correct functioning and to protect data from other processes that must be protected. Thus the system calls allow user processes to request service from the kernel, service that it would not otherwise be allowed to do (for security reasons). The interface to the system calls is through a special instruction called a "trap". This instruction switches the system into "system" mode and calls a kernel routine which looks at the parameters passed by the user function and figures out which internal operating system function to perform. Upon completion, the kernel places any return values in the appropriate place, switches back to "user" mode, and returns control to the process. Alternatively, if the request requires a lot of time to complete (as with a disk read), then the calling process may be suspended until the request has completed and other processes may be allowed to run.

3. Suppose we have a scheduling algorithm that favors processes that have used the least CPU time in the recent past.

a) Why will this algorithm favor I/O-bound processes and yet not permanently starve CPU-bound processes?

It will favor I/O bound processes because they tend to wait a long time before running. When they run, they will have received no CPU recently and will therefore have the highest priority. However, if a CPU-bound process waits long enough, it will also have had no CPU recently and will also have high priority.

b) Give a formula for priority that would result in such a scheduling algorithm. Assume that you have a function *cpu_usage(int i)* that returns the total CPU usage (up to the current time) of process *i*. Note that by calling this at time *t* and then calling it again at time *t*+1, and subtracting the two values, you can find out how much CPU was used in the interval (*t*, *t*+1).

```
// Note: low number = high priority
int last_cpu[MAX_PROCS];
int current_cpu[MAX_PROCS];
int priority[MAX_PROCS];

// At each timer interrupt:
for(int i = 0; i < num_ready_processes; i++) {
    last_cpu[i] = current_cpu[i];
    current_cpu[i] = cpu_usage(i);
    recent_usage = current_cpu[i] - last_cpu[i];
    priority[i] = priority[i]/2 + recent_usage;
}
```

c) Suppose we also want to account for processes with different importances, where more important processes should get more CPU. In other words, while favoring processes that have used the least CPU in the recent past, a process that is twice as important should get twice as much CPU. Give a new formula for priority that does this.

```
int importance[i]; // Appropriately initialized

// At each timer interrupt:
for(int i = 0; i < num_ready_processes; i++) {
    last_cpu[i] = current_cpu[i];
    current_cpu[i] = cpu_usage(i);
    recent_usage = current_cpu[i] - last_cpu[i];
    priority[i] = priority[i]/2 + recent_usage/importance[i];
}
```

4. Briefly describe these schedulers and the schedules they produce:

a) First Come First Served

Non-preemptive scheduler that services jobs or processes in the order in which they arrived in the ready queue. This scheduler is fair and does not suffer from starvation, but can result in poor responsiveness for I/O-bound processes.

b) Shortest Remaining Time Next

This preemptive scheduler services jobs in order of their remaining processing time, always choosing to run the process that needs the least amount of time to complete. It provably minimizes overall turnaround time, but may starve processes expected to take a long time if shorter processes keep entering the queue.

c) Round Robin

This preemptive scheduler rotates through the ready queue giving equal amounts of CPU time to each process. It is ultimately fair, does not suffer from starvation, but does not have especially good response time or turnaround time.

d) Priority

This is a very general scheduling mechanism depending upon the definition of priority. Priority can be static or dynamic, user/developer-specified or system determined, etc. In general, the scheduler will always execute the process with the highest priority (sometimes designated by the lowest number). The resulting schedule depends entirely on the definition of priority.

5. Deadlock

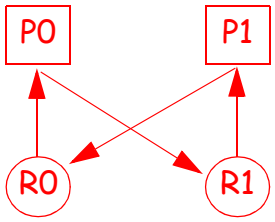
a) Name the four conditions that must hold for deadlock to occur

1. Mutual Exclusion
2. No preemption
3. Circular Wait
4. Hold and Wait

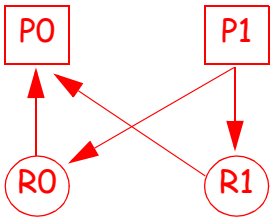
b) Name the four strategies for dealing with deadlock

1. Deadlock Avoidance
2. Deadlock Prevention
3. Detection and Recovery
4. Do nothing

c) Draw a resource allocation graph with two processes and two resources where the two processes are deadlocked.



d) Draw a resource allocation graph with two processes and two resources where both resources are allocated but the processes are not deadlocked



6. Given four page frames and the memory reference string 06514505365523235226, show what will be in each page frame after each reference using the following page replacement algorithms. Circle each page at the point that it is brought into memory because of a page fault.

a) LRU

	0	6	5	1	4	5	0	5	3	6	5	5	2	3	2	3	5	2	2	6
P0	0	0	0	0	4	4	4	4	4	6	6	6	6	6	6	6	6	6	6	6
P1		6	6	6	6	6	0	0	0	0	0	0	2	2	2	2	2	2	2	2
P2			5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
P3				1	1	1	1	1	3	3	3	3	3	3	3	3	3	3	3	3

b) FIFO

	0	6	5	1	4	5	0	5	3	6	5	5	2	3	2	3	5	2	2	6
P0	0	0	0	0	4	4	4	4	4	4	5	5	5	5	5	5	5	5	5	5
P1		6	6	6	6	6	0	0	0	0	0	0	2	2	2	2	2	2	2	2
P2			5	5	5	5	5	5	3	3	3	3	3	3	3	3	3	3	3	3
P3				1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	6

c) Optimal (assuming you know the whole reference string in advance)

	0	6	5	1	4	5	0	5	3	6	5	5	2	3	2	3	5	2	2	6
P0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3	3	3	3	3
P1		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
P2			5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
P3				1	4	4	4	4	4	4	4	4	2	2	2	2	2	2	2	2

7. Suppose that your memory management hardware supports neither reference bits nor modified bits, but does support read-only/read-write bits and valid/invalid bits. Explain how you can use the existing hardware to implement a software soluton that will emulate the functionality of the other bits. Your solution should be as efficient as possible.

The problem is that you want to know when a page has been referenced or modified. The general solution is to set pages as invalid even when they are actually valid, or to set them as read-only even when they are actually read-write. Then, when you get a page fault, you can set the appropriate bits in software and allow the access to continue.

Specifically, to emulate the reference bit, do the following. When a page is loaded into memory, set the valid bit to invalid and indicate somewhere in software that it is actually valid. Then, when you get a page fault because the hardware thinks that an invalid page has been accessed, the software can check the table and see if the page is really valid. If it is, then if the access is a read you can set the reference bit in software and if the access is a write you can set both the reference and modify bits. If the access is a read, you can then set the valid bit to valid, but set the read/write bit to read-only. Then when a write occurs, you will get a page fault and you can set the modified bit. In both cases, if the page really is valid, then you don't have to actually fetch the page. You can just return to the process and allow it to continue executing.

8. I/O Performance

a) Given a PCI bus capable of supporting data transfers of up to 500 MB/sec, and a non-DMA device controller that interrupts the CPU every time an 8-bit character is ready so that the CPU can read that character, what is the maximum number of bytes/second that we can transfer from this device? How long does it take to transfer 512 bytes?

Hmmm, I think I see why the TAs didn't like this question. It looks like it is incomplete. It depends on how fast the CPU can respond to an interrupt and read the data.

b) Now suppose that the device has a DMA controller capable of sending or recieving a 32-bit word every 100 nsec. A response takes equally long. how fast does the bus have to be to avoid being a bottleneck?

32 bits every 100+100 nanoseconds

= 4 bytes/200 nanoseconds

= 2 bytes/100 nanoseconds

= 20 MB/second

c) Suppose that the computer can read or write a memory word in 10 nsec. Also suppose that when an interrupt occurs, all 32 CPU registers plus the Program Counter and PSW are pushed onto the stack. What is the maximum number of interrupts per second this machine can process?

34 words have to be written. Then, the new PC has to be copied in. So there are a total of 35 I/O operations, each of which takes 10 nanoseconds = overhead of 350 nanoseconds = $350*10^{-9}$ seconds.

The maximum number of interrupts that can be handled in a second = 1/(350*10^-9) = 2.85 * 10^6. So, the machine can handle 2.85 million interrupts per second.

9. Describe how to implement a file system consistency checker that verifies that only free blocks are marked free, only allocated blocks are marked allocated, and each allocated block is allocated to exactly one file.

```
int blocklist[NUM_BLOCKS];
#define DONTKNOW 0
#define FREE 1
#define ALLOCATED 2

// Start by assuming everything is free
for each block in the blocklist {
    blocklist[i] = DONTKNOW;
}

// Next, go through the list of free blocks and mark each one as FREE
for each block in the free list that is marked FREE
    blocklist[i] = FREE;

// Next, go through the file system and mark each block that appears in
// a file to ALLOCATED
for each directory
    for each file in that directory
        for each block in that file
            switch(blocklist[i])
                case DONTKNOW: // This is the good case
                    blocklist[i] = ALLOCATED;
                    break;
                case FREE: // This means that it is both FREE and ALLOCATED!
                    dosomething(i);
                    break;
                case ALLOCATED: // This means that it is allocated to two files!
                    dosomethingelse(i);
                    break;
                }

// Finally, go through the blocklist and see if any are still DONTKNOW
// This is a less severe problem, and they should probably just be marked free
for each element i of blocklist
    if(blocklist[i] == DONTKNOW)
        blocklist[i] = FREE;

// Note that the above algorithm doesn't handle subdirectories
// However they are handled the same as the main directory
// This is a good example of the need for a recursive function call
```

10. In most file systems, an update to a block of a file is written back to the same block on disk. Some modern high-performance file systems such as LFS just write the updated block of data into a nearby free disk block. Briefly explain how such a system would work and describe at least one advantage and one disadvantage of this strategy.

This kind of file system works by writing updated disk blocks into any nearby (near the current position of the read/write head) free block. When they do this, they must also update the inode for the file to indicate the new location of the data block. The inode itself may also then be written to a nearby location.

An advantage of this algorithm is that seeks, a major source of delay in a disk-based storage system, can be minimized because the data can be written to a nearby location instead of seeking back to the original location.

A disadvantage is that you must constantly update the inode to keep track of the new location of the data. Any time the data is changed, its location may also change, necessitating an inode update. The constant rewriting of data will also fragment the disk, because data is being moved all of the time. This may decrease read locality at the same time that it effectively increases write locality.

11. Suppose that you have a UNIX file system where the disk block size is 1KB, and an inode takes 128 bytes. Disk addresses take 32 bits, and the inode contains space for 64 bytes of data (a recent optimization), 8 direct addresses, one indirect, one double-indirect and one triple-indirect (the rest of the space in the inode is taken up with other information such as ownership and protection). An index block is the same size as a disk block. How much space (including overhead) do files that are: One (1) byte long, 1025 bytes long, 65536 (64KB) bytes long, and 1048576 (1MB) bytes long require? Hint: it may help if you draw a picture of how inodes are used to locate the blocks making up a file.

Disk blocks are 1024 bytes
Inodes are 128 byte
Disk addresses are 4 bytes
Inodes contain space for 64 bytes of data, 8 direct addresses, 1 indirect, 1 double-indirect, and 1 triple-indirect
An index block is 1024 bytes

A single indirect pointer can point to 1024/4 = 256 disk addresses = 256 KB data.
A double indirect pointer can hold 256 * 256 = 65536 disk addresses = 64 MB data.
A triple indirect pointer can hold 256 ^ 3 disk addresses = 16 GB data

So,

1 byte file : inode stores the data : so 128 bytes (inode size)

1025 bytes : inode + 1 data block : 128 + 1024 = 1152 bytes

65536 bytes : inode + 8 direct data blocks + 1 indirect pointer block + 56 data blocks : total (128 + 65K bytes) = 66688 bytes

1 MB : inode + 8 direct data blocks + 1 indirect pointer block + 256 data blocks + (note: 760 K are still left) 4 double indirect pointer blocks of overhead + 760 data blocks : total : 128 + 1029 KB = 1053824 byte

12. Based on your knowledge of Unix and Windows and the things we have discussed in this class, describe two realistic ways that someone might be able to get another person's login information (username and password) for one of the systems on this campus without just being told the information. Note: do not simply list the names of known attacks - explain how they might actually work. Be as specific as possible.

There are many ways. Here are a few:

1. One way is to fool them into thinking they are logging in, when they are really typing their password into a program that you have written. I can think of two ways to do this:

A. Write a trojan horse program and leave it where someone will run it. For instance, write a version of *ls* that makes it look like the person was logged out and asks them to login by entering their user name and password, then really logs them out so they have to log back in. When they type the information, have the program email it to you. Leave this version of *ls* in a shared directory that many people may be accessing. Sooner or later someone will do *ls* there, running your program.

B. Log into the machine and run your fake *ls*. Then leave the machine alone. Sooner or later someone will want to use that machine. They will sit down and type their login information into what is apparently a login screen.

2. Snoop on the network and look for responses to password requests generated when people do a remote login (without ssh) onto a different machine.

3. Copy the password file into a different location. This file is readable, so you can read the usernames. The passwords are encrypted, but the encryption scheme is widely published. Many people use poor passwords, including dictionary words. Write a program that takes a copy of the dictionary and encrypts every word using the standard algorith. Compare these encrypted words to the encrypted passwords in the file. If you find a match, then you know which word was encrypted to create that password.

NOTE: Do not do any of these. You will get into very serious trouble if you are caught.